本次分享的内容是MoE，Mixture of Experts，中文翻译成混合专家模型。也是因为最近从去年年底开始DeepSeek v3推出之后就把大模型推理的成本降到了一个很低的程度，所以我也是从DeepSeek的Moe 架构梳理，把整个MoE部分做一个比较宏观上的梳理，因为涉及很细节的比如loss function换了公式为什么生效了，或者说一些超参数的调参，这些东西比较经验性，需要真正在训练中跑结果，所以特别细节的部分我的了解也很有限，也欢迎大家一起探讨，如果有不对的地方也请大家指正。

实际上这个MoE名词会给一部分人造成误解，主要是因为目前我们聊的混合专家模型已经和最一开始提出的混合专家模型发展出了很大的不同，我个人理解，和最近一些网上提出的说法结合起来来看，混合专家模型其实是一个比较拟人的说法，更加通俗的说法可以叫做稀疏激活Transformer，也就是Sparsely Activated Transformer，这个其实才是MoE模型的本质。最近DeepSeek和Kimi同时在2.18发表了两篇论文，都是关于这个思想的。DeepSeek发布的论文题目叫做NSA，Native Sparse Attention；Kimi发布的论文题目是MoBA，Mixture of Block Attention。

今天我的分享内容主要从以下七篇论文说起，除去DeepSeek发布的三篇与MoE架构有关的论文之外，我还选取了另外四篇在MoE发展过程中起到很重要作用的论文。这些论文的引用数也全部都在几千甚至大几千，在提出的当时，都是对MoE的思想产生了一些开创性的影响。

这里，我引用香港科技大学的一篇论文A survey on mixture of experts中给出的近年来MoE发展的阶段，可以看到从2023年开始MoE就开始迅速发展，几乎所有的耳熟能详的大模型都有MoE版本，而且这些列出的都是已知的开源模型，据传闻ChatGPT 4也是一个MoE模型，进入2024年之后，DeepSeek就开始推出了自己的MoE模型。

第一篇分享的论文是Adaptive mixtures of local experts, Neural Computation'1991，本地自适应专家模型。这是大多数MoE论文都引用的最早的一篇文章，发表于1991年，作者中有两个大家熟知的大佬：Michael Jordan 和 Geoffrey Hinton。在这个文章开始就提出了那时候训练模型时，Back Propagation已经遇到或者可预见的一些问题，比如当模型变得很大，不同的子任务模块会相互影响，导致训练困难。由此带来的影响就是模型泛化很差，很难训练。

这时，我们可以使用多个子模型，在文章中给了一个定义也就是专家，expert，去学习，使用一个门网络（gating network）来决定每个数据应该被哪个模型去训练，这样就可以减轻不同类型样本之间的干扰。

如图就是这篇论文提出的架构，提出了两个概念，第一个expert network，专家网络，本质上是一个前馈网络，逻辑上一个专家网络擅长处理一类专项的子任务，所有专家都接受相同的输入，来做特定计算处理，产出不同的输出。同时训练一个Gating network，门控网络，其实就是控制每个专家网络输出的权重，用于计算每个专家的重要程度。最终的输出结果就是每个专家计算的结果乘他这个专家的权重，再吧所有的专家输出求和得到整个网络的输出。

这个思路已经是很有前瞻性，但是这样的架构还是存在一定的问题，因为所有的专家都会参与计算，所以这个网络的计算量仍然是不小的。所以论文提出了自己的改进，鼓励各个专家相互竞争，最终赢家通吃，只留下一个某个子任务的专家。所以损失函数变成了单独计算loss而不是加权求和之后再计算loss，这样每个专家都会相互比较独立。

我们要纠正一个观点，这些expert并不是预先训练好的，而是再训练的过程中，逐渐收敛出来的，某些FFN就自动会收敛成为某个子任务的专家，所以会存在不同的专家在做同样的工作。所以这个论文提出了让联合网络竞争，让竞争网络联合的观点，也是很大的进步。

Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer，这是Google Brain，也就是DeepMind那个公司在2017发表的论文。从文章标题上就可以看出来它的背景和目的——希望做出极大的神经网络。这篇文章提出了 Sparsely-Gated Mixture-of-Experts layer ，声称终于解决了传统 conditional computational 的问题，在牺牲极少的计算效率的情况下，把模型规模提升1000多倍。这个文章做出了一个极大的改善在于，提出了MoE稀疏的概念。因为1991年的论文思想，存在一个比较大的问题，每一次计算都会有很多专家参与计算，Google Brain 在这个论文里提出，只需要最多两个专家参与计算即可，只需要极少数的专家参与推理，其余的权重全都是0，大大减少了计算量。通过这种方式，在激活参数很少的情况下，可以把一个模型做得很大很大。

另一个建设性的思路是把gating network，也就是routing，从sample-level细化到token-level。我个人对于MoE系统的理解就像是一个很大的三甲医院，里面分了很多很多科室，每个科室还有很多专业不同的医生专门看某一种病。在这种超大规模的医院出现之前可能就是一个赤脚医生，但是负责几乎所有的内科外科，那么想要让一个赤脚医生能达到很高的水平，这个赤脚医生可能要学很多很多年，但是如果医生只专注一个方面，那有可能能做得非常好，而且学习时间也会短些，同时对于患者来说也能更好对号入座。但是这个理解存在一个误区，这个理解过于宏观了，实际上MoE的模型分配是每个token分给不同的专家，而不是一个sample分配给某一个专家。

传统的 gating network 就是一个softmax，相关度高的专家分配到更高的权重。论文提出了一个新的gating公式，加入了稀疏性，通过TopK实现稀疏性。

另外一个问题是MoE架构的另一个重点问题，也就是负载均衡问题。在实验中发现，不同 experts 在竞争的过程中，会出现“赢者通吃”的现象：前期变现好的 expert 会更容易被 gating network 选择，导致最终只有少数的几个 experts 真正起作用。因此作者额外增加了一个辅助loss，来缓解这种不平衡现象，也就是把一个批次的全部权重加起来，最后这个loss会加到总体的loss中，鼓励不同的experts一起发挥作用。用这个架构思想，这个RNN-LSTM的翻译模型参数训练到了137B，这在当时算一个很大的网络，训练出来算是很大的成功。

以上两篇论文奠基了MoE早期的架构。2021年，Google发布GShard，这是第一次将MoE的思想拓展到Transformer上。具体的做法是，把Transformer的encoder和decoder中，每隔一个（every other）的FFN层，替换成position-wise 的 MoE层，使用的都是 Top-2 gating network。

从架构图中我们可以发现，MoE其实就是将Transformer中的FFN层替换成了MoE-layer，其中每个MoE-Layer由一个gate和若干个experts组成。这里gate和每个expert都可以理解成是nn.linear形式的神经网络。

不同token代表的含义不一样，因此我们可以用不同expert来对它们做解析。除了训练上也许能达到更好的效果外，MoE还能帮助我们在扩大模型规模的同时保证计算量是非线性增加的，因为每个token只用过topK个expert，不用过全量expert，这也是我们说MoE-layer是稀疏层的原因。最后需要注意的是，在之前的表述中，我们说expert是从FFN层转变而来的，这很容易让人错理解成expert就是对FFN的平均切分，实际上你可以任意指定每个expert的大小，每个expert甚至可以>=原来单个FFN层，这并不会改变MoE的核心思想：token只发去部分expert时的计算量会小于它发去所有expert的计算量。

除此之外，这篇论文还提出一个对后续大规模模型训练很重要的思路，就是分布式多设备训练和推理，把MoE也分配到不同的设备上进行运算。

GShard还在Balancing方面提出了几个新的方法：

强制每个expert处理的tokens数量在一定范围内

强制两个专家来处理token，第一个softmax最高值的专家是确定的，第二个专家采用一定随机性，按照权重随机

同时辅助损失函数也是为了缓解赢家通吃的问题

Switch Transformer也是2021年由谷歌提出的模型，但是论文在2022年才发表，它给MoE模型换了一个名字，它进一步简化了Gating，再次提升了计算效率。同时Swith Transformer 在论文中提到其设计的指导原则是——尽可能地把Transformer模型的参数量做大。Switch Transformer在当时做到了1.6万亿参数，到目前为止仍然是已知参数最大的神经网络模型。它把MoE不再局限在端到端翻译，而是作为一个Foundation model。

跟其他MoE模型的一个显著不同就是，Switch Transformer 的 gating network 每次只 route 到 1 个 expert，而其他的模型都是至少2个。这样就是最稀疏的MoE了，因此单单从MoE layer的计算效率上讲是最高的了。同时由于GShard提出的MoE也会跨设备训练，所以不同的专家层需要在分布式系统中训练，通信开销变得越来越大。

因此Google提出了一个概念叫做Expert capacity。这个参数来限制每个专家在一个批次中可以接受多少token用于运算。这个参数又一个公式给出。其中这个公式中有一个超参数capacity factor，容量因子，用于调整专家的容量。Google也在大量训练中找到了一个比较优秀的值，几乎不会影响performance，但是可以减少很多开销。

Switch Transformer一方面在做简单路由，另一方面实现了高效稀疏路由。最理想的负载均衡结果是，当有N个专家的时候，每个专家被分配到的概率应该是均匀的，但是这样的分布是离散的，如果是离散的就没法反向传播，所以Google想了一个可微的loss function，引入了概率，这是一个logits分布，这样就可以正常反向传播。

限制了专家处理的token数量之后，就会存在有些token可能没有专家处理了，Google引入了一个变量Dropout-rate来控制这部分token是否直接丢弃。发现Dropout-rate在0.4左右时，模型的表现很好。

deepseekv1的这篇论文的标题叫做Towards Ultimate Expert Specialization in Mixture-of-Experts Language Models，意思就是要让每个专家超级专业。DeepSeek MoE主要是想解决两个问题，

知识混合：此前很多MoE的工作用到的expert太少（通常是8或者16），导致很多expert不是真正的专家，而是杂糅了各种知识的混合体，所以并不能发挥expert的专业作用；

知识冗余：不同专家训练的token很可能存在共同的知识，导致不同的expert会共享一些共通的知识，从而出现冗余，而模型整体容量有限，冗余就造成MoE模型整体性能下降。

针对以下问题，DeepSeek提出了一个更新的架构。

一个针对的改进是Fine-Grained Expert Segmentation，中文理解为细粒度专家分割，就是在整体参数量不变的情况下，将FFN在latent维度切分成更多expert，同时，保证计算量不变的情况下可以激活更多专家。相应地，在保持计算成本不变的情况下，可激活更多细粒度的专家，以实现激活专家组合的更高灵活性。细粒度专家分割使得多样化的知识能够被更细致地分解，并更精确地学习到不同的专家中，每个专家将保持更高的专业化水平。

如下公式所示，deepseek-MoE相当于把expert的个数从 N 个提升为 mN 个，同时激活的专家数量也从 k 个提升为 mk个。

另一个针对的改进是共享专家隔离Shared Expert Isolation：将某些专家隔离出来，作为始终激活的共享专家，旨在捕获不同上下文中的共同知识。通过将共同知识压缩到这些共享专家中，可以减轻其他路由专家之间的冗余，这可以提高参数效率，确保每个路由专家专注于不同方面而保持专业化。

如图就是DeepSeek MoE的结构图，黄色的柱状图代表softmax计算的结果，选取最高的几个路由，同时保持了Transformer里的残差连接，这样当实际MoE层数很大的时候也不会特别难训练。

此外，MoE里针对大模型的训练中，负载不均衡的情况给了一个新的定义叫做routing collapse，中文可以翻译成路由崩溃或者叫路由塌缩。主要是两种情况：

模型总是选择某些特定的expert，导致其他的大部分专家没有得到充分训练；

并且自从GShard提出在不同的device上训练之后，设备间通信带来的开销越来越大。如果不同的专家位于不同的设备上，通信代价会超过计算瓶颈；所以针对这两个问题，deepseek又分别设计了Expert-Level Balance Loss 和 Device-Level Balance Loss。

先看专家级负载的loss，计算公式如图

其中阿尔法1是超参数，用来调节和主网络loss的权重，T是专家要处理的全部token数量，1代表indicator function，指示函数就是被路由到这个专家的概率。其实这个公式和之前的Switch Transformer给出的负载均衡公式相比，改进的点就在于在f’的前面加上了一个系数，这个系数实际上是为了消除不同激活专家数量对于损失函数的影响，使得最终的损失值在不同激活专家的情况下都能保持一个稳定的区间范围，不会浮动太大。

设备级别的辅助损失函数很明显就是为了专家并行方案时，token过于集中在某些卡专家而设置的。将MOE层所有的专家进行分组，分D组 ，每一组放在不同的设备上，则为了平衡不同设备间负载均衡问题。

在MoE论文的基础上，DeepSeek很快推出了V2，其实改动并不大，在辅助损失函数方面，又提出了一个辅助函数。也就是通信损失函数，用于保证设备之间的通信是比较平衡的，思路还是类似的，用一个系数来抵消不同的设备的影响。

除此之外，有点类似dropout的策略，deepseek提出了token dropout的策略，亲和度得分最低的token将会直接被丢弃，但是有10%的token保证永远参与计算。

在V3中，Deepseek将gating function从softmax改为了sigmoid，论文中没有直接给出原因，我自己的猜测和网上的一些参考资料来看，大概率是为了节省计算资源的，毕竟gating function要的并不是softmax具体的得分，而只是得分的排名，其实用sigmoid也是可以比较的。在sigmoid 后有一个normalization，应该是能实现softmax的效果的。

为每个专家引入一个偏置项 bi，并将其添加到对应的亲和度分数 Si，t中，用于确定 Top-K 路由。其中si，t是原始的亲和力分数，bi是偏置项，在训练过程中，实时监控每个专家的负载情况。如果某个专家过载，则减少其偏置项 bi​；如果某个专家负载不足，则增加其偏置项。调整速度由超参数 γ 控制。

这个调整是因为虽然辅助损失函数可以平衡不同的路由，但是带来的额外计算负担越来越大，DeepSeek就是为了减少计算的损耗，加快训练的过程。

设备受限的路由这一点是沿用了V2中的设备受限路由，最多选择 M 个设备中的专家进行路由，通过这个策略可以做到通信与计算的重叠。

不再使用token丢弃策略在V2中虽然使用了多种辅助的负载均衡损失函数，但实际训练中都没有达到很好的负载均衡，而在V3中训练中通过引入偏置项来控制路由的策略，以及序列级别的token辅助负载均衡损失有效的使得整个训练过程负载均衡，从而不再寻token丢弃的策略。